

PATENT

Atty. Docket No.: ROC920030145US1 (IBM/257)
Confirmation No. 7590

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant: Paul Reuben Day et al. Art Unit: 2167
Application No.: 10/660,166 Examiner: Michael Pham
Filed: September 11, 2003 Atty. Docket No.: ROC920030145US1
For: METHOD AND SYSTEM FOR DYNAMIC JOIN REORDERING

DECLARATION OF PAUL REUBEN DAY UNDER 37 CFR §1.131

Mail Stop Amendment
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

I, Paul Reuben Day, hereby declare and state:

1. I am an inventor of the above-identified U.S. Patent Application.
2. Prior to April 21, 2003, I and my co-inventor, Brian Robert Muras, conceived of a concept of a database engine and a system running a database engine utilizing a dynamic join reordering feature to change the order of two or more join operations while a query is executing. With our concept, a database engine starts execution of a query with an initial join order setting but monitors the execution of the query to determine whether the initial join order or some other join order would provide better runtime performance. If another join order would provide better performance, then the database engine can change the join order during query execution and complete the query using the new join order.
3. In particular, we conceived of an apparatus, a program product, and a method for monitoring a query involving a plurality of join operations during runtime, including running the query according to a first join order, and concurrent with running the query, collecting performance statistics about each of the join operations.

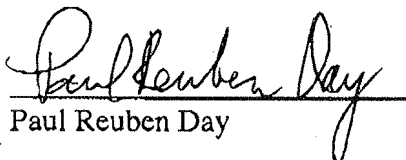
4. We also conceived of an apparatus, a program product, and a method for optimizing a query join order during runtime, where the query involves a plurality of join operations, including running the query according to a first join order, collecting statistics about each of the join operations concurrent with running the query, and based on the collected statistics, selecting a preferred join order, while running the query, such that the query continues to run according to the preferred join order.

5. Prior to April 21, 2003, I and my co-inventor submitted an invention disclosure form to our employer, International Business Machines Corporation. Our improved apparatus, program products and methods were described in this form (with portions thereof redacted), and a copy of this form is attached hereto as Exhibit A.

6. From the period prior to April 21, 2003, until the filing of the patent application on September 11, 2003, I and my co-inventor were diligent in constructively reducing our invention to practice. During this time period, we met with inside counsel at International Business Machines Corporation to explain our invention, participated in telephone calls with the outside counsel assigned to prepare the patent application, provided supplemental materials to outside counsel to assist in preparing the patent application, reviewed a draft of the patent application prepared by outside counsel, supplied comments and suggested revisions to outside counsel, and reviewed and executed a final draft of the patent application.

7. The statements made herein of my own knowledge are true, and all statements made on information and belief are believed to be true; further, these statements are made with the knowledge that willful false statements and the like are punishable by fine or imprisonment, or both, under Section 1001, Title 18 of the United States Code, and that such willful false statements may jeopardize the validity of the above-referenced application or any patent issuing thereon.

Respectfully submitted,


Paul Reuben Day

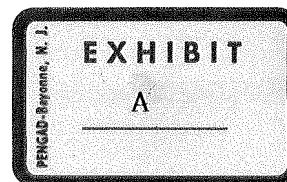
4-20-2006
Date



Disclosure

Prepared for and/or by an IBM Attorney - IBM Confidential

Created By Brian Muras On
Last Modified By Brian Muras



Required fields are marked with the asterisk (*****) and must be filled in to complete the form .

*Title of disclosure (in English)

Autonomic Join Query Fan-In Detection and Dynamic Join Reordering

Summary

Status	Submitted
Final Deadline	
Final Deadline Reason	
*Processing Location	Rochester
*Functional Area	select (2R6) 2R6 - SG - Data-Centric Development - David R. Nelson
Attorney/Patent Professional	Steve Roth/Rochester/IBM
IDT Team	select Rich Diedrich/Rochester/IBM Eric Barsness/Rochester/IBM Michael Branson/Rochester/IBM Bill Berg/Rochester/IBM Thomas J Eggebraaten/Rochester/IBM Scott Gerard/Rochester/IBM Greg Leibfried/Rochester/IBM Gary Ricard/Rochester/IBM Bill Schmidt/Rochester/IBM Blair Wyman/Rochester/IBM Steve Roth/Rochester/IBM
Submitted Date	
*Owning Division	select SG
Incentive Program	
Lab	
*Technology Code	601
PVT Score	

Inventors with a Blue Pages entry

Inventors: Brian Muras/Rochester/IBM, Paul Day/Rochester/IBM

Inventor Name	Inventor Serial	Div/Dept	Inventor Phone	Manager Name
> Muras, Brian R.	754448	7T/HRGA		Foster, Dennis C.
Day, Paul R.	709414	7T/HRGA		Foster, Dennis C.

> denotes primary contact

Inventors without a Blue Pages entry

IDT Selection

Attorney/Patent Professional	Steve Roth/Rochester/IBM
IDT Team	Rich Diedrich/Rochester/IBM
	Eric Barsness/Rochester/IBM
	Michael Branson/Rochester/IBM
	Bill Berg/Rochester/IBM
	Thomas J Eggebraaten/Rochester/IBM
	Scott Gerard/Rochester/IBM
	Greg Leibfried/Rochester/IBM
	Gary Ricard/Rochester/IBM
	Bill Schmidt/Rochester/IBM
	Blair Wyman/Rochester/IBM
	Steve Roth/Rochester/IBM

Response Due to IP&L

*Main Idea

1. Background: What is the problem solved by your invention? Describe known solutions to this problem (if any). What are the drawbacks of such known solutions, or why is an additional solution required? Cite any relevant technical documents or references.

One of the primary tasks of the query optimizer is to determine what the optimal join order is for a given join query. A poor join order can result in a very poor performing query whereas a good join order may result in excellent performance of the query. The query optimizer is thus tuned to try to find this optimal join order. However, the general solution of always finding the optimal join order for every possible query is not tractable due to the general nature of SQL, the different data of every user, and different system environments of every user.

Good query performance basically comes from minimizing I/O as the various tables and indexes are scanned to return the records the user is interested in. The goal then is to apply the most selective selection early on in the join processing so that records which the user is not interested in will be discarded at the earliest possible time. That is, the records will be discarded with minimal processing. Many types of selection which minimize I/O are detectable through various optimization strategies; however, selection acquired through "**Fan-In**" is not detectable at optimization time.

Fan-In is implicit selection which occurs because a record in one file which is joined to a second file does not find a match therein. (Conversely, Fan-Out is where a record from one file joins to many records in a second file).

For example, consider the below query which joins X to Y and X to Z. For this particular example, let us assume that X will be the file in the primary join position, and the optimizer has a decision of whether to join to Y next or to Z next. If the optimizer joins, X -> Y then the join will fan-out to 20 records since each value in X.j1 joins to two values in Y.j1 (10 records in X * 2 average duplicates in Y = 20 joined records) whereas, X -> Z will fan-in because for every record in X.j2, there are no records in Z which match (10 records in X times 0 matches = 0 joined records).

Thus, for performance reasons we would be better off joining X->Z first followed by Y because the query would fan in to zero records via X->Z and not need to touch Y at all. However, if we join X->Y followed by Z, we must touch Z 20 times (because of X->Y fan-out) and throw the record away thereafter because the Z join condition does not match.

Note that in this case, current optimization strategies do not easily apply. Two common strategies which

can not help here, for example, are the **average number of duplicate join values**, which is the same in both case for Y and Z (ie two), and the **number of records** in both files Y and Z, which are the same (ie twenty). This problem of detecting the best join order before the query is submitted is not always possible as this example shows. Thus, the need for this invention which allows the database engine to monitor and analyze the query while it is running and dynamically adjust the join order as appropriate.

```
select * from X, Y, Z
where X.j1 = Y.j1 and X.j2 = Z.j2
```

Table X

j1 j2

1	1
1	2
1	3
1	4
1	5
1	6
1	7
1	8
1	9
1	10
2	1
2	2
2	3
2	4
2	5
2	6
2	7
2	8
2	9
2	10

Table Y

j1

1
1
1
1
1
1
1
1
1
1
2
2
2
2
2
2
2
2
2
2

Table Z

j2

1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10

2. Summary of Invention: Briefly describe the core idea of your invention (saving the details for questions #3 below). Describe the advantage(s) of using your invention instead of the known solutions described above.

As the database engine is running the join, it will keep statistics on how many records it has processed from the "join from" file (X in above example), how many records were discarded, and which "join to" file discarded them. By logging these observations, the database engine will have statistics by which to make an informed decision about dynamically adjusting the join order.

In the above example, suppose that we join in the order X->Y->Z. As the database engine is processing the join, via the statistics the engine is now keeping, it observes that Z is responsible for causing much fan-in, whereas Y is causing fan-out. At some threshold, the engine can then decide to switch the join order, and begin joining in the order X->Z->Y. In this example, this would result in Y never needing to be probed again.

With these "in process" join statistics, slightly more complex algorithms can easily be developed. For example with large files, there are many standard statistics/probability algorithms which could be applied. For example, if the above X, Y and Z files contained millions of records, we could use the first *i* records

processed in X as a sample of the population of records of X, and we could randomly choose to probe Y or Z and log the results. When the sample size of X had grown to a large enough value to be statistically confident to some threshold of error we could know the best file join order to lock in. A heuristic could also be applied to learn the best sample size of X and determine a learned threshold. For example, query run times that are just below the current threshold, and query run times that just meet the threshold can be compared. Depending on the differences in these timings, the threshold can be adjusted incrementally either up or down. It is anticipated that the heuristic will settle to a different threshold on different machines and for different data bases, because there are many variables that could affect the timings that can not otherwise be taken into account with a hard coded setting in the algorithm.

Additionally, the algorithm can easily be flexible to accommodate switching join order more than once based on the history of the join. It should be noted that the overhead of switching the join order would be negligible.

3. Description: Describe how your invention works, and how it could be implemented, using text, diagrams and flow charts as appropriate.

The point which allows this algorithm to work and select the correct records is once the query starts fetching records, it must hold the left most join file in the query fixed (i.e. X above). Thus, only secondary joins can be considered for reordering. Within the fixed left most file (ie X), we process records in an orderly and sequential fashion either through an index or table scan. For each record in X, we compare it's "join-from" values to the "join-to" values in secondary join positions. The order we process the join-to values from the secondary positions, from a functional point of view, is arbitrary, but from a performance perspective, it can be a large difference.

On the iSeries platform, the query optimizer would need to identify which files of the query are eligible for dynamic reordering, and which are not based on join type (such as distinct, inner, left outer, right outer, exception etc). Additionally, the algorithm would only consider reordering in certain cases where the decision to order X->Y over X->Z is considered arbitrary within some threshold using traditional join optimization metrics. For example, if the average join duplicates of Y is 2 and the average duplicates of Z is 100 then we would not consider X->Z over X->Y, but if we were presented with average duplicates of 2.7 in Y and 3.1 in Z for similarly sized files, we may consider the decision somewhat arbitrary. In this case, we would probably start the join order as X->Y->Z but tell the database that it has the freedom to reorder X->Z->Y if it detects a certain threshold of fan-in.

Then the database engine (slicdb on the iSeries) could add support to keep statistics such as the count of records processed, the number of fan-ins per file, and/or the average fan-out. Support could also be added to run the secondary join cursors in any specified order as opposed to a hardcoded order in the current implementation. Then a new autonomic manager function could monitor the statistics and update the specified join order as appropriate for the "in process" join. Thus, support for Autonomic Join Query Fan-In Detection and Dynamic Join Reordering would not be difficult to implement.